FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Information System for Draw Competitions
Student:	Hlib Yarovyi
Supervisor:	Ing. Marek Suchánek
Study program:	Informatics
Branch / specialization:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2022/2023

Instructions

Draw competitions are a common phenomenon both in commercial and personal environments. Various companies want to gain attention from potential customers by creating a draw competition where people enter codes from products, answer some questions, guess some product-related information, and so on. Similarly, friends or groups of people may do their own competition. The common aspects are random drawing of winner(s) and assigning prizes. The goal of this thesis is to create an information system to support this activity:

- Analyse the domain of draw competitions (both commercial and personal).
- Research briefly the existing solutions.
- Design and implement a custom solution as a web information system that will help to configure and manage such competitions, select winners, and provide API for integrations.
- Test and evaluate the solution.

Electronically approved by Ing. Michal Valenta, Ph.D. on 12 October 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Information System for Draw Competitions

Hlib Yarovyi

Department of Software Engineering
Supervisor: Ing. Marek Suchanek

May 10, 2022

Acknowledgements

I would like to express my gratitude to my supervisor, Ing. Marek Suchanek, for his support for my thesis. Also, I would like to thank my parents for all the support they have given me during my studies. And of course, I want to thank myself for the diligence and immense work.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 10, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Yarovyi, Hlib. *Information System for Draw Competitions.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Tato bakalářská práce popisuje proces návrhu a implementace REST API pro systém podporující losovací soutěže. Navržená implementace používá architekturu mikroservis a několik návrhových vzorů souvisejících s mikroservisy, například API Gateway nebo Load Balancer. REST API je navrženo pro integrace s jinými službami, obstarává manipulaci dat a vyhodnocení vítěze losovací soutěže. Navžený systém je implementován, otestován a také zdokumentován pomocí OpenAPI.

Klíčová slova Informační systém, Návrh, implementace, Microservices, Java, Spring, API, API Gateway

Abstract

This bachelor thesis describes the process of designing and implementing a REST API for the drawings competitions system. The proposed implementation uses the microservices architecture and several microservices-related design patterns such as API Gateway or Load Balancer. The REST API is designed for integrations with other services and manages data manipulation and winner selection for drawing competitions. The designed system is implemented, tested and also documented using OpenAPI.

Keywords Information System, Design, Implementation, Microservices, Java, Spring, API, API Gateway

Contents

1 Introduction **1**

2 Objectives **3**

3 Analysis **5**
3.1 Domain of commercial draw competitions 5
3.2 Domain of personal draw competitions 5
3.3 Requirements 5
3.3.1 Must have 6
3.3.2 Should have 6
3.3.3 Could have 7
3.3.4 Will not have 7
3.4 OntoUML model 8
3.5 Existing solutions 10
3.5.1 Amazon fun zone 10
3.5.2 E-bay auction 10
3.5.3 Wheel of names 11

4 Design **13**
4.1 Microservices 13
4.1.1 Discovery Server 14
4.1.2 Identity Provider Service 15
4.1.3 Drawings Manager Service 16
4.1.4 Service per each Drawing type 16
4.1.5 Support Services 17
4.1.6 API Gateway Service 17
4.1.7 Swagger UI Service 17
4.2 Technology Stack 17
4.2.1 GitLab 17
4.2.2 MySQL database 18

4.2.3 Spring Boot . . . 18
4.2.4 Spring Security . . . 18
4.2.4.1 Authentication . . . 19
4.2.4.2 Authorization . . . 19
4.2.4.3 Servlet Filters . . . 19
4.2.5 Spring Cloud Gateway . . . 20
4.2.6 Netflix Eureka . . . 20
4.2.7 JSON Web Token . . . 21
4.2.7.1 JSON . . . 21
4.2.7.2 Token . . . 21
4.2.8 OpenAPI . . . 21
4.2.9 Postman . . . 22
4.2.10 JUnit5 . . . 22
4.2.11 Test Containers . . . 22

5 Implementation **23**
5.1 Drawings Manager Service . . . 23
5.2 Docker-compose . . . 23
5.3 Codebase structure . . . 24
5.4 Common Library . . . 25
5.5 Limits service . . . 25
5.6 Config service . . . 25

6 Testing **27**
6.1 Postman . . . 27
6.2 JUnit5 . . . 28
6.3 Swagger UI . . . 28

7 Evaluation and future steps **29**
7.1 Completed requirements . . . 29
7.2 Future steps . . . 29
7.2.1 Set up HTTPS . . . 29
7.2.2 Distributed tracing . . . 30
7.2.3 Kubernetes . . . 30

Conclusion **31**

Bibliography **33**

A Acronyms **35**

B Contents of enclosed CD **37**

List of Figures

1.1 16th Century Italian Lotteries 1

2.1 Software Development Life Cycle 3

3.1 OntoUML Model User part 8
3.2 OntoUML Model Drawing part 9
3.3 OntoUML Model Bank part 10
3.4 Ebay auction for epdu 11
3.5 Wheel of names page 12

4.1 Architecture monolithic vs microservices 13
4.2 Netflix Eureka server workflow 14
4.3 Login operation workflow 15
4.4 Design: Quiz Drawing update 16
4.5 Security filter chain 19

5.1 Docker-compose configuration for database 24
5.2 Config Server code 26
5.3 Config Server configuration 26

6.1 Postman environment 27

Chapter 1

Introduction

The first lotteries and prize draws began a very, very long time ago. According to the assumptions of some scientists, the appearance of hoaxes is associated with Ancient Greece, but most of the history of occurrence is associated with Ancient China or Rome.

The chronicle [1] says that from 100 to 44 BC. the ancient Roman rulers Nero and Augustus loved to organize lotteries during the holidays, where slaves and property were played. The first public lottery was also organized to raise funds for the city needs of Rome: the repair of roads, bridges and buildings. In addition, during major holidays, free lotteries were organized for local residents. People were given "papers of happiness", and a few lucky ones received cash prizes. No wonder the lottery (from the Italian lotto) in Italian means "Fate".

Figure 1.1: 16th Century Italian Lotteries

These days draw competitions are a common phenomenon both in commercial and personal environments. Various companies aimed at increasing

sales, loyalty or visitor traffic by creating a draw competition where people enter codes from products, answer some questions, guess some product-related information, and so on. Similarly, friends or groups of people may do their own competition. The common aspects are random drawing of winner(s) and assigning prizes.

After some research, a lot of videos of people running private drawing contests have been found on the YouTube platform. They are trying to draw some things. If people want to win, they have to buy tickets for a small price. Moreover, they do not use any software for this. All payments are made through the messenger. If people are playing this game in such a bad environment, it would be better if there was some safe platform for it. This will save the owner time and money, and participants will be less nervous.

Chapter 2

Objectives

This paper is intended to analyse upon the implementation the domain of commercial and personal draw competitions. During the creation of the thesis, the traditional process of software development in software engineering was followed [2].

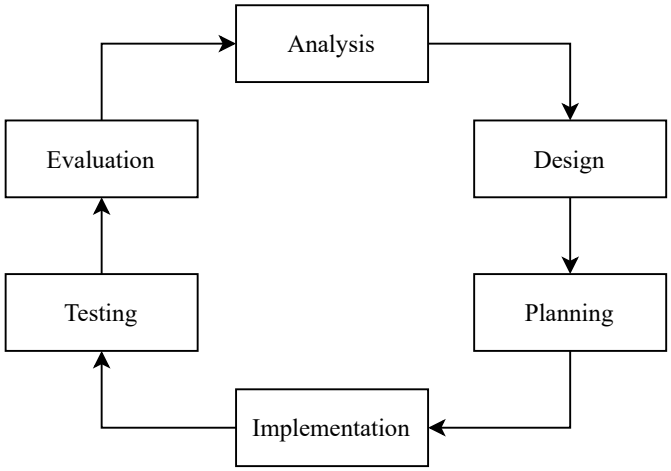

Figure 2.1: Software Development Life Cycle

The thesis has following objectives:

- Analyse the domain of draw competitions (both commercial and personal).
- Research briefly the existing solutions.
- Design and implement a custom solution as a web information system that will help to configure and manage such competitions, select winners, and provide API for integrations.

- Test and evaluate the solution.

Chapter 3

Analysis

3.1 Domain of commercial draw competitions

Free draws and prize competitions can be run for commercial gain and can be used when promoting a product. It is great opportunity to generate interest in some product or service. Marketers are increasingly using online prize draws to boost interaction and generate leads, especially during peak sales periods.

3.2 Domain of personal draw competitions

A personal drawing can be used in some variations. First of all, if a person wants to sell some item for a good price, they can create a drawing ticket and specify how much each ticket will cost. In addition, he will specify how many tickets must be sold before a winner is determined. As a result, the winner will have some expensive product at a very low price and great emotions because he won it. Besides that, the owner will sell the item at full price.

Secondly, group of friends could buy some product for common use. Unfortunately, this item must be held by someone and friends start arguing. Here drawing application can help. One of the solutions is free drawing, where they can randomly choose a person who will receive the prize. Second solution is quiz drawing, where each of friends can create some questions and add them to system. Further, each of person will solve quiz and winner will hold prize.

3.3 Requirements

Before starting implementation an analysis was carried out using the MoSCoW method. The MoSCoW method is a four-step process for determining which project requirements will yield the most profit.

3.3.1 Must have

- Support User authorization and authentication.

  Sign in/up user to system.

  Check if user has permissions to use API.

- Support Quiz type drawing.

  Process Quiz information(prize details, deadlines, questions, etc.).

  Add questions to system.

  Provide quiz questions to user.

  Process quiz answers from user.

  Evaluate total result(win/lose).

- OpenAPI specification.

  Specify all exposed URLs.

  Specify all request and response data.

  Specify all request fields in requested data.

3.3.2 Should have

- UI/UX.

  Design UI and UX for any platform.

- Support ticket type drawing.

  Process drawing information(prize details, amount of tickets, price per ticket, etc.).

  Add prize to system.

  Provide ticket purchase service.

  Provide random selection of a winner.

  Provide prize to winner.

- Support free type drawing.

  Provide free assigning to drawing.

  Provide random selection of a winner.

  Provide prize to winner.

- Mock bank system.

  Structure for init implementation of bank system.

  Should have money wallet.

Mock for connection to real bank.

Provide purchase of local currency without payment.

- Support API for 3d party users.

  Provide registration of new users through 3d party companies API.

3.3.3 Could have

- Support money transactions.

  Create service to connect to real bank.

  Provide purchase of local currency with real money payment.

  Create bank account for application.

- Drawing chat for participants.

  Chat window in drawing details where participants can discuss their chances to win.

- Exposed host for public.

  deploy application to public host.

- Licence service for 3d party.

  Provide fixed-term license purchases for companies that want to integrate the drawing application into their systems.

- NFT as prize.

  Create NFT storage.

  Support prize as NFT.

  Store NFT prizes in storage.

- Blockchain wallet support.

  Provide blockchain wallet registration instead of system one.

3.3.4 Will not have

- Perfect UI.

- Kubernetes implementation.

  Create Docker containers for each service

  Move all Docker containers to Kubernetes cluster as nodes

- Deliver installable image.

  Provide package with application

  Provide library with interfaces

3.4 OntoUML model

Full OntoUML model can be found on CD.

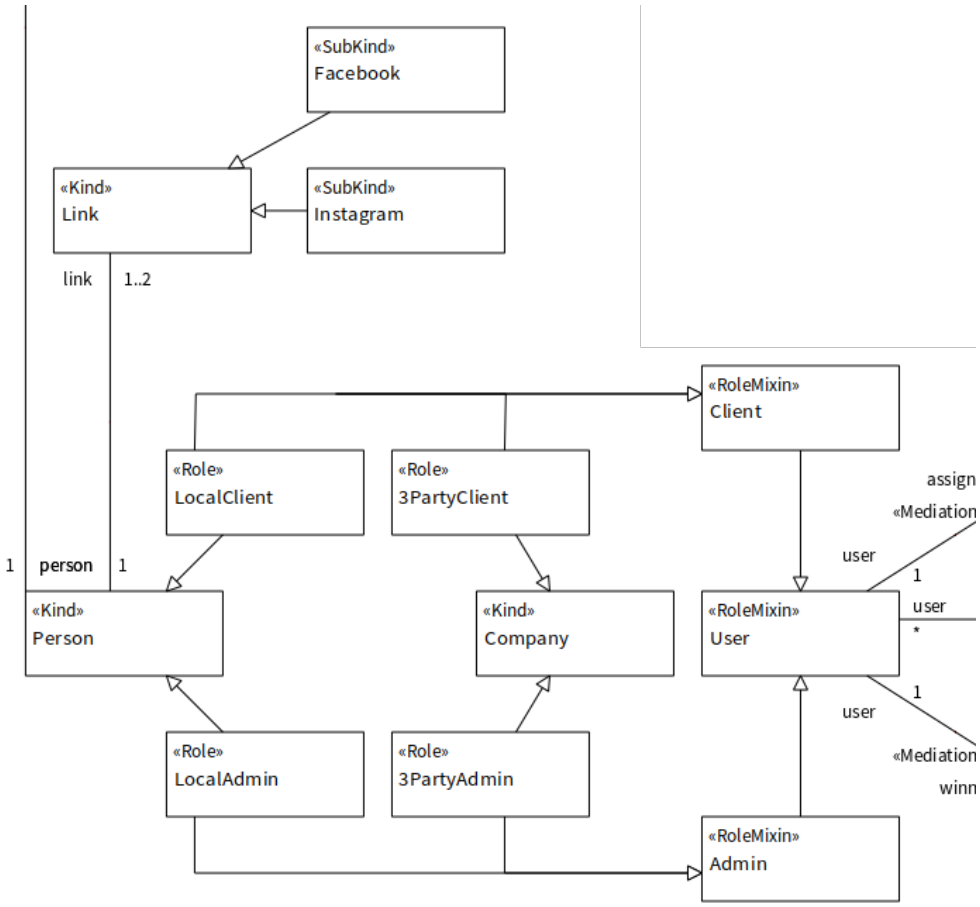

Figure 3.1: OntoUML Model User part

User can be presented as basic person or some Company. Basic person can fill in some contact links as Instagram or Facebook. Furthermore, 3d party Company can create some drawing for their employees and register them in system with special rules. Will be provided functionality to login through 3d party identity provider API. Company admin should configure connection to API.

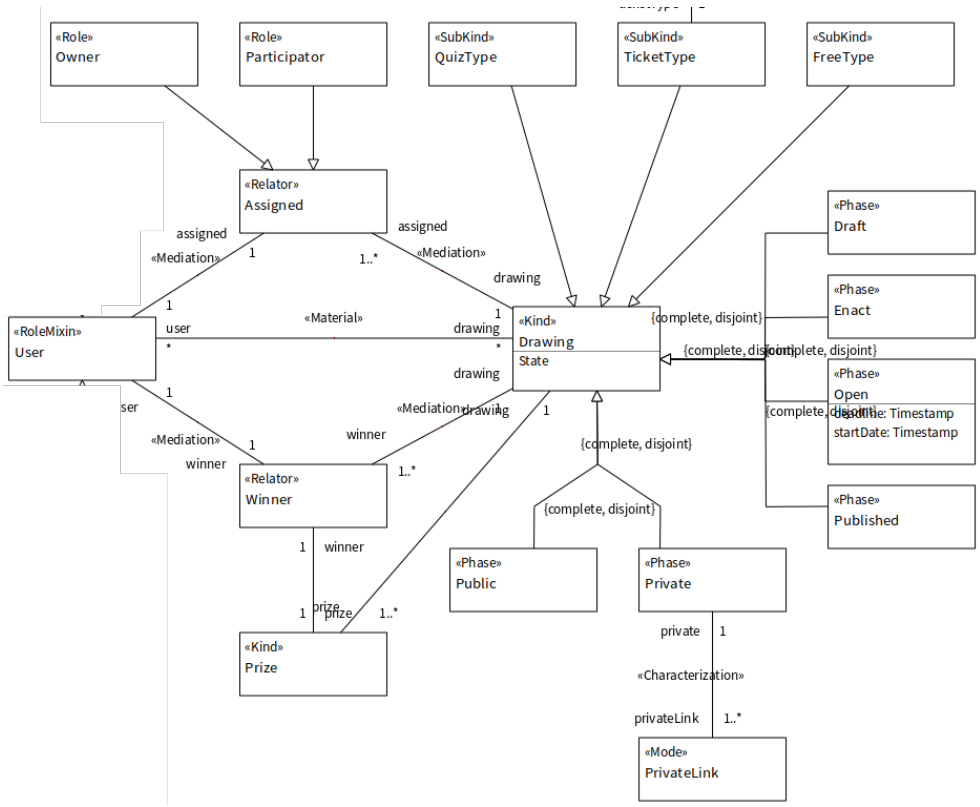

Figure 3.2: OntoUML Model Drawing part

Each drawing has multiple states and must be assigned by at least one User(Owner who created). Drawings has public and private phases, it is mean that private drawing you can access only by link. Moreover, each drawing can have from 1 to many winners. Consequently, there should exist 1 to many prizes. Drawing divided into three groups: Quiz, Ticket and Free. Owner can choose which type of drawing he wants.

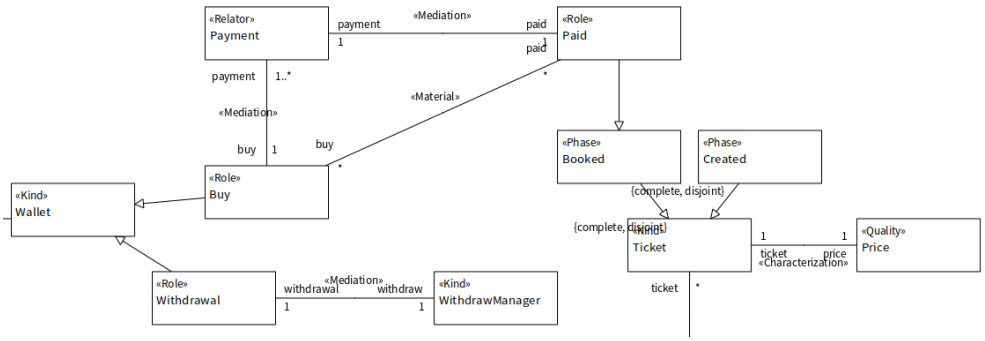

Figure 3.3: OntoUML Model Bank part

For Ticket type drawing there exist price for each ticket and some states: Booked and Created which show us if ticket was assigned to any person. Assignment goes by paying the ticket price. Payment goes from Wallet which holds current financial state of user in our system. User can pay for something in the system or get money back by withdraw operation.

3.5 Existing solutions

3.5.1 Amazon fun zone

Amazon fun zone [3] is an official Amazon-run website where you can discover daily quizzes, fascinating new games, and puzzles. There you can win prizes everyday. As there are millions of people plays in amazon fun zone and the number prizes are low. It was not possible to give rewards to everyone. There is only one or two winner if they are giving good amount in Amazon pay or some expensive item. Unfortunately, it exist on mobile app only.

Moreover, Amazon fun zone looks like a scam application. Users has not trust to it. It you search for it you will find huge amount of question about safety of the project. Fortunately, this opens up space for competition and development of other quiz related products.

3.5.2 E-bay auction

EBay's auctions use a fixed-time semi-sealed-bid modified second-price format. The highest bidder at the end of the auction wins, regardless of when their bid was put, even if it was received at the last second and didn't appear on your screen before the countdown reached zero, but the price is set second in terms of participant size.

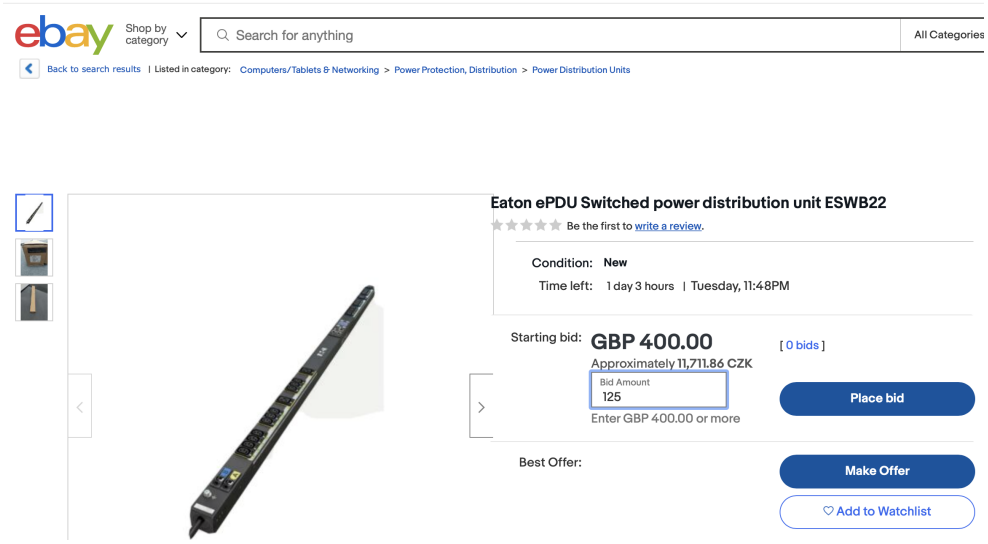

Figure 3.4: Ebay auction for epdu

This is usually the lowest bidder's highest bid plus one bid increment. Less if the winning bid is not a complete step higher than the underbid, such as in the case of a previous tie. More if the reserve in the reserve list needs to be satisfied. If there is only one bidder and no reserve, the seller sets the beginning bid amount.

3.5.3 Wheel of names

The main goal of making a name wheel is to pick a person at random from a group of people [4]. This ensures that everyone has an equal chance of being chosen.

For example, the selection could be for selecting a student to answer a question in class. A wheel spin could also be used to determine who wins a competition or a prize giveaway.

Figure 3.5: Wheel of names page

This name picker wheel can also be used for a variety of other purposes. Unfortunately, all names must be filled in manually and will not be saved in the future. Furthermore, the implementation is not the highest quality.

Chapter 4

Design

4.1 Microservices

Microservices [5] was chosen as the architectural style for designing the application. It's a type of architecture that aims to break down monolithic apps into smaller pieces. The application is built as a collection of self-contained services, each with its own set of responsibilities.

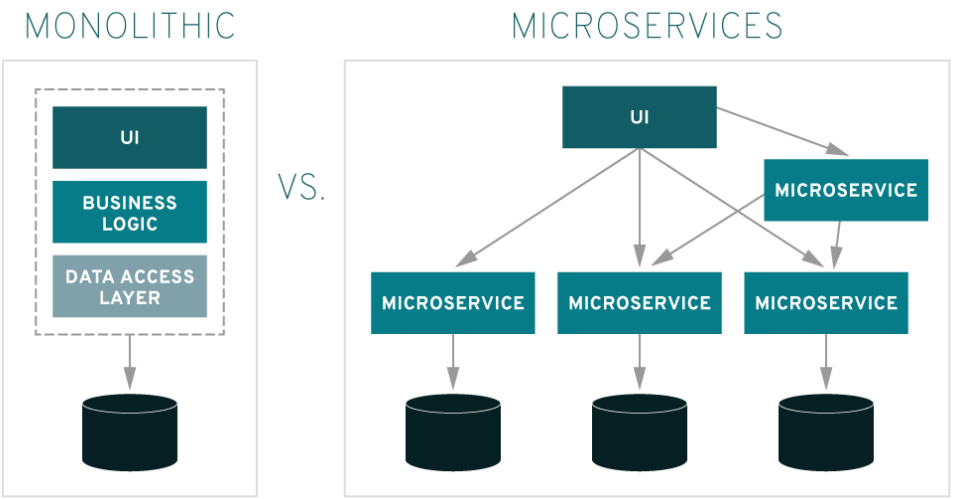

Figure 4.1: Architecture monolithic vs microservices

Breaking an application down into smaller autonomous fragments makes it easier to build and maintain. Depending on the demands of each service, it can be built, launched, and managed individually, and it can use a variety of programming languages, technology, and software environments.

Moreover, multiple teams can be assigned for different services and divide associated tasks. New components can be added without requiring downtime and redeployment of the entire system. Because each module of an application has a smaller codebase, it's easier to release, scale, deploy, and test different services.

The fact that each service can be written in a different language or technology gives opportunity for CI/CD team to choose the most suitable stack of technologies for each service. Services can also be distributed across numerous servers, reducing the performance effect of more resource-intensive components.

In case of monolithic architecture that considered to be a traditional way of building applications. A monolithic application is made up of one indivisible unit. A client-side user interface, a server-side program, and a database are typically included in such a system. All functions are handled and served from a single location. Monolithic apps typically have a single huge code base and lack modularity. Developers use the same code base when they wish to update or replace something. As a result, they make changes to the entire stack at the same time.

4.1.1 Discovery Server

Discovery server allows services to find and communicate with each other without hard-coding the hostname and port. The service registry, with which each service must register, is the only 'fixed point' in such an architecture. To interact with this fixed point, all clients must implement a specific logic – register themselves as clients on discovery server.

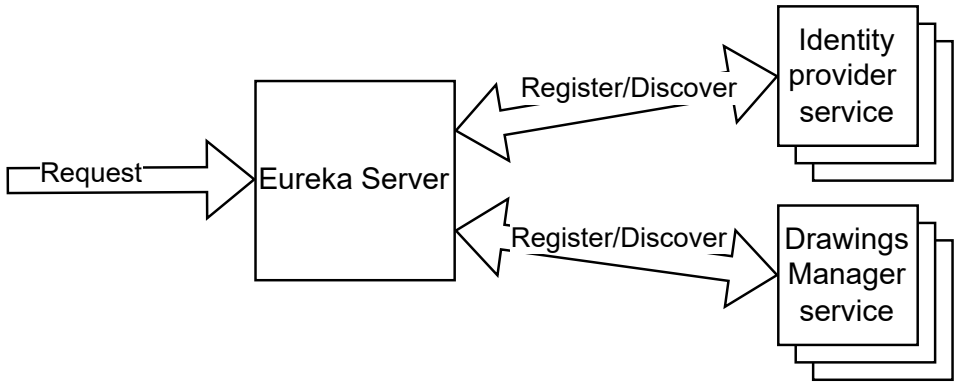

Figure 4.2: Netflix Eureka server workflow

The application makes use of Netflix Eureka [6], which allows any client to act as a server and send its status to a connected peer. In other words, a

client obtains a list of all connected peers from a service registry and uses a load-balancing mechanism to make all subsequent requests to other services. Load balancing is the process of distributing traffic among various instances of the same application.

Advantages:

- Handles all instances.
- No need to specify exact host and port on client side.
- Reduce the load on your web servers.
- Optimize traffic.

Disadvantages:

- If the eureka server goes down, then the whole application goes down.
- Weighted Response Time.

4.1.2 Identity Provider Service

Main technologies are Spring Boot [7] and Spring Security [8].

Service can provide identity to user or service. Identity is provided in the form of a JSON Web Token. Response contains access token and refresh token. Access token – contains username, user roles and expiration date.

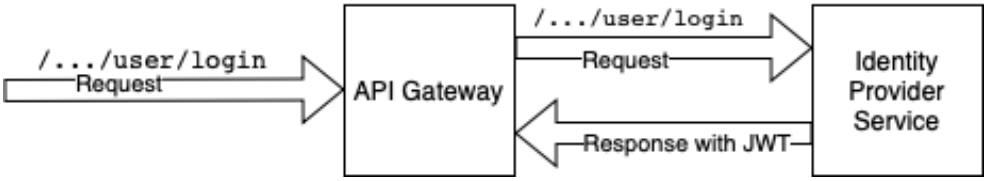

Figure 4.3: Login operation workflow

Moreover, identity provider service can authorize and authenticate user. User can sign in and sign up to system via API requests.

Also, each service must have JSON Web Token to authenticate themselves. The first action a identity provider service takes when receiving a request is authentication of a request. It must have header with valid JWT which contains username of registered service and valid roles for system service. Thus we limit the work of the provider with unverified services directly.

4.1.3 Drawings Manager Service

Main technology is Spring Boot [7].

This service handles requests around drawings. The service must create new drawing and manage its common information. It also sends a request to the identity provider to authenticate the user.

Furthermore, drawing manager cooperates with each service of draw competition. So, it is somehow used as gateway. It must validate user information and forward the request to the appropriate service.

All common drawings information stored in MySQL database [9].

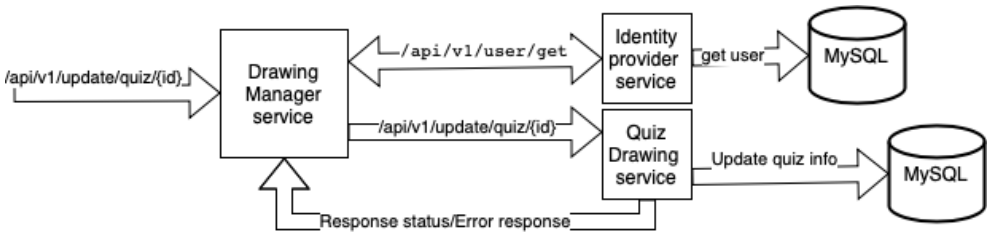

Figure 4.4: Design: Quiz Drawing update

On the model presented workflow around request which updates quiz information.

Firstly, Drawings Manager Service receives request from different service.

Secondly, it start user validation through Identity Provider Service. The application must ensure that the user has sufficient rights to update this quiz drawing.

Thirdly, manager service forward request to quiz drawing service. This service tries to update everything and send response back to drawings manager service. The response can be empty with a CREATED(201) status, or an Error response describing the problem that occurred.

4.1.4 Service per each Drawing type

Main technology is Spring Boot [7].

This services should handle all type specific operations.

- Quiz type drawing should handle requests to create question, create prize, provide list of questions to solve, check answers and more.
- Ticket type should provide tickets, randomly select winner.

4.1.5 Support Services

This services should handle all supportive operations.

- Prize service – save prizes per drawing and provide them when needed.
- Mail Sender service – should send email notifications, application announcements to users.
- Wallet service – should store users balance and manipulate with it.

4.1.6 API Gateway Service

Main technologies are Spring Boot [7] and Spring Cloud Gateway [10].

An API gateway is a solution for managing APIs that lies between a client and a group of backend services. An API gateway serves as a reverse proxy, accepting all API calls, aggregating the numerous services required to fulfill them, and returning the appropriate result.

In case of a drawings application, the gateway must add a header with service JWT [11] authorization to every request, since all services require an authenticated request.

As a result, the API Gateway will receive a request, add a header with an authorized access token, send a request to the appropriate service through discovery server, send response back.

4.1.7 Swagger UI Service

Main technologies are Spring Boot [7] and OpenAPI [12].

Swagger is a set of open-source tools for designing, building, documenting, and consuming REST APIs based on the OpenAPI Specification [12]. Swagger's most important tools are: Swagger Editor is a browser-based editor for writing OpenAPI specifications. It's a useful tool for seeing and interacting with API resources without having to worry about the implementation.

In case of a drawings application it will be provided as a UI. Swagger UI turns OpenAPI specifications into interactive API documentation.

4.2 Technology Stack

4.2.1 GitLab

GitLab [13] is a web-based Git repository that offers open and private repositories, as well as issue tracking and wikis. Was used as hosting platform for version control and issue planning tool.

4.2.2 MySQL database

MySQL [9] is an open source SQL-based relational database management system. A relational database is one that divides data into numerous independent storage locations called tables and associates them with each other using keys. A table is a collection of related data, and is made up of columns and rows. A key is a unique numerical ID number. With key it is possible to link the data from these tables together allowing you to manipulate and mix the data in various tables as needed.

4.2.3 Spring Boot

Spring Boot [7] is a Java-based open source framework for developing microservices. It's used to create self-contained, production-ready Spring applications. Configure Java Beans, XML settings, and Database Transactions in a customizable fashion. Makes dependency management easier. It also maintains REST endpoints and offers robust batch processing. Everything is auto-configured in Spring Boot. Aids in avoiding extensive XML configuration in Spring, making it easier to design production-ready Spring applications, reducing development time, and allowing the application to function independently.

4.2.4 Spring Security

Spring Security [8] is a framework that focuses on servlet filters that help you easly add authentication and authorization to your web application. The actual value of Spring Security, like all Spring projects, is in how readily it can be modified to meet specific requirements. It works well with frameworks such as Spring Boot and standards like OAuth2 and SAML.

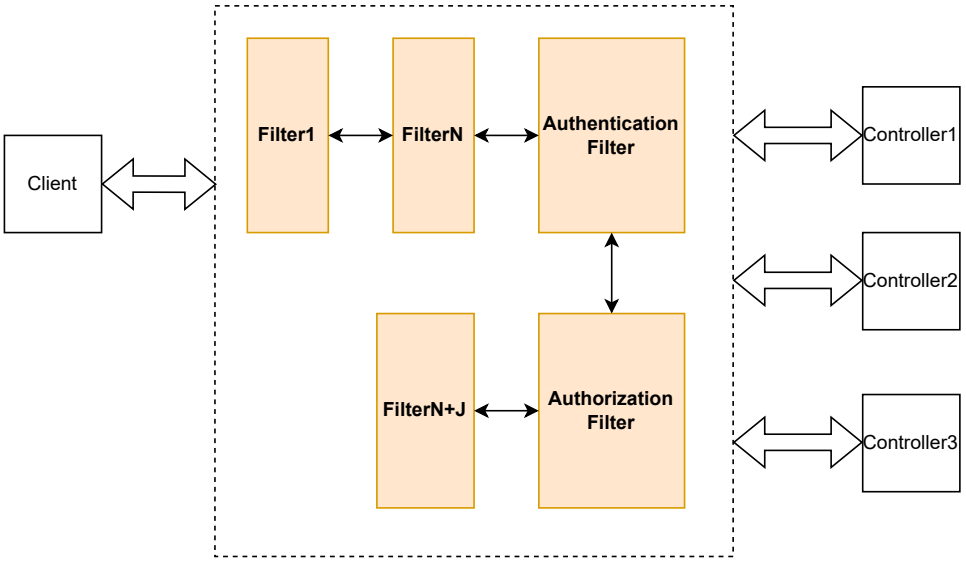

Figure 4.5: Security filter chain

4.2.4.1 Authentication

Authentication is the process of recognizing a user's identity. It's the process of connecting a set of identifying credentials with an incoming request. The credentials provided are compared to those stored in a database containing the information of authorized users. Before any other code is authorized to begin, the authentication process runs at the start of the application, before the permission and throttle checks.

The credentials are usually in the form of a username and password. Password kept private and only known by the user and the system.

4.2.4.2 Authorization

Authorization is the process of giving the user permission to access a specific resource or feature. It determines what resources a user has access to.

In secure environments, authorization must always come after authentication. The system determines what information the user is permitted to access after successful user authentication. Before the system grants users access to the requested resources, they must first verify their identities.

4.2.4.3 Servlet Filters

A filter is an object used to intercept the HTTP requests and responses of application. By using filter, we can perform operations before sending the

request to the controller or sending a response to the client. Mostly filters used to validate the data coming from the client to server.

Spring Security provides a number of filters by default. In spite of this, you can create a new filters to use them in the filter chain. Internally, Spring Security maintains a filter chain, with each filter having a specific responsibility and filters being added or withdrawn from the configuration based on which services are required. Because there are dependencies between the filters, the sequence in which they are applied is crucial.

4.2.5 Spring Cloud Gateway

Spring Cloud Gateway [10] is a Java and Spring-based library for creating API gateways. An API gateway allows you to remove the complexity from the client, shifting the responsibility from the user to the server. An API Gateway is a single point of entry for a group of microservices. External clients cannot access the microservices directly; instead, they must go through the application gateway. This means that the client only has to know how to communicate with the gateway. It makes no difference if the backend services migrate, go down, or become unreliable as long as the gateway is prepared to handle them.

Spring Cloud Gateway aims to provide a simple yet effective approach to route to APIs and address cross-cutting concerns like security, resiliency, and monitoring. Because the Spring Cloud Gateway is focused on routing requests, it sends them to a Gateway Handler Mapping, which indicates what should be done with requests that match a certain route. It is intended to sit between a requester and a resource that's being requested, where it intercepts, analyzes, and modifies every request. That is, queries can be routed based on their context.

Spring Cloud Gateway is a non blocking API. It follows that when you use Spring Cloud Gateway, no incoming request is ever blocked. When using a non-blocking API, a thread is always ready to handle the incoming request. These requests are then processed asynchronously in the background, and the response is returned once they have been completed.

4.2.6 Netflix Eureka

It is a lookup service where microservices has role of clients and can register themselves and discover other registered microservices. When a client microservice registers with Eureka [6], it gives metadata like host, port, and health indicator, which other microservices can use to find it. Each Netflix Eureka client can work as a server at the same time, replicating its state to a linked peer. In other words, a client obtains a list of all connected peers from a service registry and uses a load-balancing mechanism to make all subsequent requests to other services.

Discovery server is another name for Eureka Server. The discovery server must be informed about the availability of a client, that is why each of clients have to send a heartbeat signal to the registry. If an instance fails to deliver a heartbeat on a regular basis, the discovery server will remove it from his registry.

This will result in a very stable ecosystem of microservices that collaborate with one another. Furthermore, we do not have to manually maintain the addresses of other microservices, which is nearly impossible to do while scaling up and down frequently.

4.2.7 JSON Web Token

JSON Web Token [11] is an open standard that allows two parties — a client and a server — to share security information. Because of its relatively small size, a JWT can be delivered through URL, POST parameter, or HTTP header. Each JWT contains encoded JSON objects, including a set of claims.

JWTs use a cryptographic technique to ensure that the claims cannot be changed after the token has been issued.

JWTs can be signed with either a secret (using the HMAC algorithm) or a public/private key pair (using RSA or ECDSA). When public/private key pairings are used to sign tokens, the signature additionally verifies that only the party with the private key signed it. Signed tokens can be used to verify the validity of the claims they contain, whereas encrypted tokens keep those claims hidden from third parties. To avoid querying a database several times, a JWT provides all of the required information about an entity. A JWT recipient does not need to contact a server to verify the token.

4.2.7.1 JSON

JSON is a text-based data transfer standard for web applications. It keeps information in an easily accessible format for both developers and computers. Any programming language can utilize it as a data format.

4.2.7.2 Token

A token is a data string that represents another object, such as an identity.

4.2.8 OpenAPI

The OpenAPI specification [12] is an API description format for REST APIs. It defines a standard, language-agnostic interface that is used to create, describe, consume, and visualize RESTful APIs and web services. An OpenAPI file allows to describe entire API specifications in YAML or JSON formats that allows both humans and computers to discover and understand the capabilities of the service without access to documentation or source code.

The main advantage of using a standard definition is that when OpenAPI specification is properly defined the third-party users can interact with and understand the service with minimal amount of implementation logic. Furthermore, documentation creation tools can leverage the OpenAPI specification to display the API.

4.2.9 Postman

Postman [14] is an HTTP request testing API client. It allows developers to easily design, share, test, and document APIs. When it comes to executing APIs, Postman is quite useful. Postman improves collaboration and simplifies each step of the API lifecycle so you can build better APIs faster.

Furthermore, Postman allows you to bundle different requests together. Users allowed to create and save simple and complex HTTP/s requests that can be simply reused over and over again, without having to remember the exact endpoint, headers, API keys, or other. This feature is referred to as 'collections,' and it aids in the organization of testing. These collections are folders that store requests and can be structured in any way the user wants. They can also be exported and imported.

4.2.10 JUnit5

In the Java ecosystem, JUnit [15] is one of the most prominent unit-testing frameworks. The JUnit 5 version includes a number of exciting new capabilities aimed at supporting new features in Java 8 and higher, as well as providing a variety of testing techniques.

4.2.11 Test Containers

Testcontainers [16] is a Java package that enables JUnit tests by providing lightweight, disposable instances of common databases and other applications that may run in a Docker container. As a result, it is possible to write self-contained integration tests that do not rely on other resources.

Any resource that has a docker image can be used in testing. Databases, web browsers, web servers, and message queues, for example, all have images. As a result, we may use them in our tests as containers.

Chapter 5

Implementation

Created as designed:

- Discovery Server
- Identity Provider Service
- API Gateway Service
- Swagger UI Service

5.1 Drawings Manager Service

This service handles requests around drawings. The service creates new drawing and manage it. It also sends a request to the identity provider to authenticate the user.

Drawing manager can create quiz drawing and handle all functionality around it. It can provide quiz information, create questioner to solve, check answers and more. Moreover, it handles all common information of drawing competition.

5.2 Docker-compose

Docker-compose [17] file configuration was created for database initialization.

```yaml
version: '3'

services:
  mysql_db:
    image: mysql:5.7
    restart: always
    hostname: dbhost
    container_name: "drawings_mysql_db"
    environment:
      MYSQL_ROOT_PASSWORD: toor
    ports:
      - '3308:3306'
    expose:
      - '3308'
    volumes:
      - db_vol:/var/lib/mysql
      - ./create-db.sql:/docker-entrypoint-initdb.d/databases-backup.sql
    networks:
      - def

volumes:
  db_vol:
    external: false

networks:
  def:
    driver: bridge
```

Figure 5.1: Docker-compose configuration for database

This configuration downloads MySQL image from docker hub. The next step, is to create docker container and deploy previously downloaded image to this container. It expose port for outside world, that is why application can use it directly as basic database. Moreover, before image deployment it creates specific volume and runs SQL init script which creates database.

5.3 Codebase structure

Each service was written on Java and stored in one GitLab repository. Moreover, Apache Maven [12] was used as build tool. So, it was decided to create a multi-module project. Root directory built from an aggregator POM that manages a group of submodules and has common dependencies for each module(service).

The submodules are regular Maven projects, and they can be built separately or through the aggregator POM. It provides easy way to compile all services by one command.

```
services ..... the directory wich contains all services
  ├─ config-server .3 pom.xml ..... maven configuration for this service
  ├─ discovery-server
  │   └─ pom.xml ..... maven configuration for this service
  ├─ drawing-gateway
  │   └─ pom.xml ..... maven configuration for this service
  ├─ drawings-manager
  │   └─ pom.xml ..... maven configuration for this service
  ├─ identity-provider
  │   └─ pom.xml ..... maven configuration for this service
  ├─ limits-service
  │   └─ pom.xml ..... maven configuration for this service
  ├─ services-common-web
  │   └─ pom.xml ..... maven configuration for library
  ├─ swagger-ui
  │   └─ pom.xml ..... maven configuration for this service
  └─ pom.xml ..... maven configuration for all services in package
└─ pom.xml ..... root maven configuration
```

5.4 Common Library

This library used for storing common helpers. Each service, which handles request, has special package in common library. This package stores DTOs and URls per each service, which has request mappings. URLs storage is really good approach to store it, because there exist no way to create invalid handler per mapping.

Moreover, the application has very smart exceptions for REST part. All common interfaces, services for exception are stored there as well.

5.5 Limits service

Provides limits per service. Limiting the number of concurrent requests avoids the remainder of the system from failing by limiting the amount of system resources that service invocations can consume.

5.6 Config service

Provides configurations to each service connected. All configuration stored in private GitHub repository. Service connects to this repository and get appropriate configuration file. The next step is to assign this configuration file to appropriate connected microservice.

```java
@EnableConfigServer
@SpringBootApplication
public class ConfigServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication.class, args);
    }

}
```

Figure 5.2: Config Server code

Config service uses Spring Cloud Config Server [18] which allows easy integration. Screenshot above demonstrates all codebase of rhis module.

```
spring.application.name=configuration-server
server.port=8888
spring.cloud.config.server.git.uri=https://github.com/****
spring.cloud.config.server.git.username=*****
spring.cloud.config.server.git.password=****

spring.cloud.config.server.git.clone-on-start=true
```

Figure 5.3: Config Server configuration

To use this framework, you need to use a specific annotation in the source code and a little extra configuration in resources package.

Chapter 6

Testing

Testing was carried out in several ways, both automated and manual. Many simple and not so bugs have been fixed through testing. Thanks to this process, the current version of the application is quite stable and predictable.

6.1 Postman

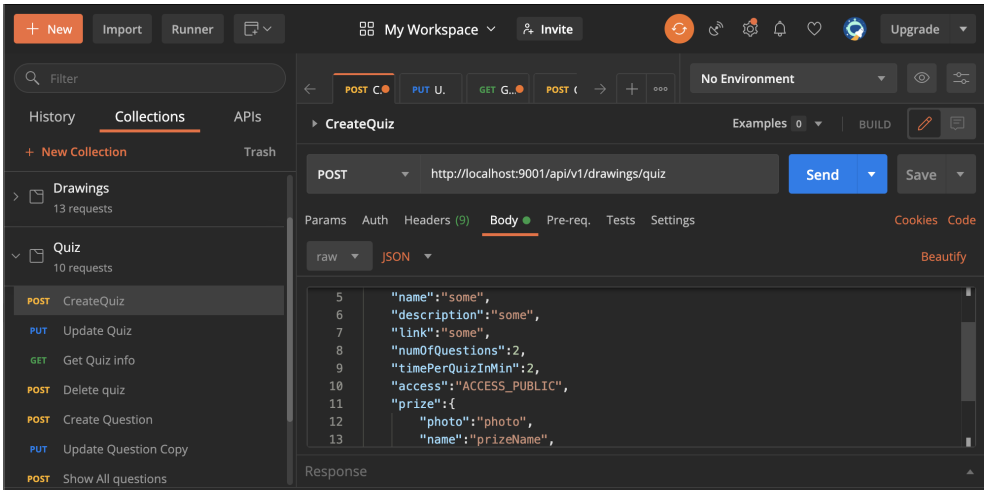

Figure 6.1: Postman environment

Postman was used as manual integration testing. It has the ability to make various types of HTTP requests (GET, POST, PUT, PATCH, etc.). Addresses to certain services have been created and saved along with data for sending. This helped not to refill the data to send each time. After sending the request, there was a check for the correctness of the response. The check was carried

out manually, it was checked if each response was correct. If the response was not within the scope of validation, then the error was looked for in the implementation.

Most of bugs were around error response. For example, when a user submits an invalid password, it should return an invalid credential error, while it returned an invalid email address error. In addition, bugs were found with the default error response. It must return custom error response and sometimes the application did not use template for it and sent with default trace message.

6.2 JUnit5

JUnit5 framework was chosen for automated testing. Integration testing was carried out. The verification consisted in sending data to a certain entry point. After that, the response was received and validated. The result was checked both positive and error.

Furthermore, integration testing does not harm the real data in the database, the Test Containers framework was used. Technology can be used to establish a temporary database within a Docker container. The real database must then be replaced with one from the container.

Some bugs was discovered during testing. For example, once the user was not found and the application did not check it. It threw a NullPointerException which broke the valid request processing. In addition, some issues were found with user IDs being stored incorrectly in the database. Thanks to Unit testing this issue was solved quickly.

6.3 Swagger UI

The Swagger UI helped to test the API gateway service. Swagger UI generated above OpenAPI documentation. It provides functionality to send requests to some servers. In the case of this application, the Swagger UI is configured to send requests to the API Gateway service. All existing requests were sent, and responses were manually reviewed.

Chapter 7

Evaluation and future steps

7.1 Completed requirements

The mandatory requirements set during the analysis phase were successfully done. In addition, pre-designed services were also implemented successfully.

The most significant completed topics:

- Support User authorization and authentication.
- Support Quiz type drawing.
- OpenAPI specification.
- Discovery Server.
- Identity Provider Service.
- Drawings Manager Service.
- API Gateway Service.
- Swagger UI Service.

7.2 Future steps

7.2.1 Set up HTTPS

HTTPS is a secured HTTP request or response. HTTPS encrypts HTTP requests and responses with TLS (or SSL), so an attacker would see a series of seemingly random characters instead of the text.

TLS implements a technique known as public key encryption, in which two keys, a public key and a private key, are shared with client devices via the server's SSL certificate. When a client establishes a connection with a server,

the two devices use the public and private key to agree on new keys, known as session keys, to encrypt further communications between them.

7.2.2 Distributed tracing

The capacity of a tracing solution to follow and observe service requests as they move through distributed systems by collecting data as the requests pass from one service to the next is known as distributed tracing. In other words, it will help to manage logs between services and it will show sequence of services which was used for each request.

7.2.3 Kubernetes

Kubernetes [19] is an orchestration tool for containerized applications. It enables the deployment of containerized microservices to be automated. This makes it easy to manage all of the application's components and microservices.

Service discovery and config server handled by Kubernetes automatically. So, it will open ability to remove some services and stop writing configuration for it.

Conclusion

The objective of this thesis was to implement the domain for commercial and personal draw competitions. In addition, the implementation was built on the basis of a Microservice architecture using some additional Design Patterns. As a result, an API was developed and documented with OpenAPI. All microservices were built on the basis of the Spring Boot framework, which simplified the creation of each service and provided many additional features for easy use of third-party technologies. Furthermore, the ground has been prepared for the easy creation of new services and their integration into the project.

As far as analysis is concerned, the MoSCoW method was very helpful in understanding architecture building. In the future, it is necessary to plan each topic in more detail.

The design process was also carried out before the start of implementation. With no experience in microservice architecture, the design was done quite successfully. Only one service was done a little differently. The design was carried out for more functionality, which will help in the future to immediately start implementing new features.

Moreover, testing of the received API was carried out. Thanks to this, a sufficient number of bugs were eliminated.

Bibliography

[1] Wikipedia. Lottery. Available from: https://en.wikipedia.org/wiki/Lottery

[2] Dooley, J. *Software Development and Professional Practice*. 2011, ISBN 978-1-4302-3802-7.

[3] Amazon. Available from: https://www.amazon.in/b?ie=UTF8&node=14351766031&ref_=nav_custrec_signin&

[4] wheelofnames.com. Available from: https://wheelofnames.com

[5] AppDynamics. What Are The Benefits of Microservices Architecture? Available from: https://www.appdynamics.com/topics/benefits-of-microservices

[6] Spring. Spring Cloud Netflix. Available from: https://spring.io/projects/spring-cloud-netflix

[7] Spring. Spring Boot. Available from: https://spring.io/projects/spring-boot

[8] Spring. Spring Security. Available from: https://spring.io/projects/spring-security

[9] Oracle. MySQL. Available from: https://www.mysql.com

[10] Spring. Spring Cloud Gateway. Available from: https://docs.spring.io/spring-cloud-gateway/docs/current/reference/html/

[11] Auth0. Introduction to JSON Web Tokens. Available from: https://jwt.io/introduction

[12] Swagger. OpenAPI Specification. Available from: https://swagger.io/specification/

[13] GitLab. GitLab documentation. Available from: https://docs.gitlab.com

[14] Postman. Introduction. Available from: https://learning.postman.com/docs/getting-started/introduction/

[15] JUnit. JUnit 5 User Guide. Available from: https://junit.org/junit5/docs/current/user-guide/

[16] North, R. Testcontainers. Available from: https://www.testcontainers.org

[17] Docker. Overview of Docker Compose. Available from: https://docs.docker.com/compose/

[18] Spring. Spring Cloud Config. Available from: https://cloud.spring.io/spring-cloud-config/reference/html/

[19] Kubernetes. Kubernetes Documentation. Available from: https://kubernetes.io/docs/home/

Appendix A

Acronyms

API Application Programming Interface

CI/CD Continuous Integration, Continuous Delivery.

DTO Data Transfer Object

Git Global Information Tracker

HMAC Hash-based message authentication code

ID Identity

JWT JSON Web Token

REST Representational State Transfer

SQL Structured Query Language

UI User Interface

URL Uniform Resource Locator

UX User Experience

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `exe` ..... the directory with executables
- `src` ..... the directory of source codes
  - `docker-compose.yaml` ..... docker-compose file to deploy database
  - `db-init.sql` ..... script to initialize database
  - `services` ..... directory with each service source code
  - `pom.xml` ..... root build configuration
- `docs` ..... the thesis documentation directory
  - `latex` ..... the directory of LATEX source codes of the thesis
  - `OntoUML Model.png` ..... full OntoUML Model
  - `thesis.pdf` ..... the thesis text in PDF format